JOURNAL OF COMPUTER AND SYSTEM SCIENCES 48, 1–8 (1994)

Three One-Way Heads Cannot Do String Matching

MIHÁLY GERÉB-GRAUS

Department of Computer Science, Tufts University, Medford, Massachusetts 02155

AND

MING LI*

Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

We prove that three-head one-way DFA cannot perform string matching, that is, no three-head one-way DFA accepts the language $L = \{x \# y \mid x \text{ is a substring of } y, \text{ where } x, y \in \{0, 1\}^*\}$. This answers the $k = 3$ case of the question whether a k-head one-way DFA can perform string matching, raised by Galil and Seiferas.

1. INTRODUCTION

The string-matching problem is defined as follows [GS]: Given a character string x, called the *pattern*, and a character string y, called the *text*, find all occurrences of x as a subword of y.

It is well known that string matching is a very important practical problem. Since the linear-time algorithms of [BM, C, KMP], there have been constant efforts to find linear-time algorithms which use less space. The best theoretical result is in [GS], where it was shown that string-matching can be performed by a six-head *two-way* deterministic finite automaton in linear time. In [GS], it was observed that a *one-way* k-head deterministic finite automaton (k-DFA) must operate in linear time, and it was asked whether string matching could be performed by a one-way k-head deterministic finite automaton (k-DFA thereafter), for some k. In [LY], it was shown that the answer is no for $k = 2$.

As the main result of this paper, we develop new simple techniques and answer

* During this research, Ming Li was supported in part by the NSF Grant DCR-8606366 at Ohio State University, by ONR Grant N00014-85-K-0445 and ARO Grant DAAL03-86-K-0171 at Harvard University, and by NSERC Operating Grants OGP0036747 and OGP0046506 at York University and University of Waterloo.

0022-0000/94 $6.00

the question for the case of $k = 3$. We actually prove a slightly stronger result which asserts that three one-way heads cannot even detect the existence of the pattern in the text, as stated precisely in Theorem 1. The current proof greatly simplified a previous version [L] of this paper. We believe our current proof will provide useful techniques for the general problem.

2. Preliminaries and Notation

A k-DFA $M = \langle \Sigma, Q, \delta, q_0, F, k \rangle$, is a deterministic finite automaton with k one-way read heads $h_1, h_2, ..., h_k$. Σ, Q, δ, q_0, F are the alphabet, the set of states, the transition function, the starting state, and the set of final states, respectively. M has a one-way read-only input tape. We assume that the input to M is of the form, $\$pattern\#text\$$, where w.l.o.g. we assume $\Sigma = \{0, 1\}$ and $pattern, text \in \Sigma^*$. The \$ signs serve as endmarkers. Initially, all k heads are on the first bit of input. At each step, depending on the current state and the ordered k-tuple of symbols seen by the heads $h_1, h_2, ..., h_k$, M deterministically changes state and moves some of the heads one position to the right. Without loss of generality, on each input all heads will eventually reach and stay at the final \$ sign. M *accepts* an input if it is in a final state when all heads reach the end. Our lower bound holds even when we allow that the heads can see each other, that is, they can detect their coincidences so that the k-DFA may change state differently if some heads meet.

ID_t of M on input I is the $(k+1)$-tuple: $(q, i_1, i_2, ..., i_k)$, where q is a state and i_j for $1 \leqslant j \leqslant k$, is the position of the jth head at time t.

Kolmogorov complexity has played an important role in lower bound proofs. Its usage in lower bound proofs was first introduced in [P, PSS]. We define Kolmogorov complexity of a string x, denoted by $K(x)$, to be the length of the shortest program that prints x (only), with respect to some enumeration of programs. A string x is *random* if $K(x) \geqslant |x|$. The conditional Kolmogrov complexity of x with respect to y, denoted by $K(x \mid y)$, is the length of the shortest program which, with extra information y, prints x; x is *random relative to* y if $K(x \mid y) \geqslant |x| - \log|y|$. We state two simple well-known facts without proofs. See, for example, [PSS, RS, M, LV1], or the survey [LV].

Fact 1. There exist random strings of each length. In fact, most strings are random.

Fact 2. If string $x = uvw$ is random, then $K(v \mid uw) \geqslant |v| - O(\log|x|)$.

For convenience, we sometimes write $x - v$ to denote the string obtained by deleting all occurrences of v from x (since we are dealing with random strings no occurrence of v will overlap with other occurrence of v for long enough v).

3. Three One-Way Heads Cannot Do String Matching

THEOREM. *No 3-DFA accepts* $L = \{\$x\#y\$ \mid x \text{ is a substring of } y\}$.

Proof of the theorem. Suppose a 3-DFA $M = \langle \Sigma, Q, \delta, q_0, F, 3 \rangle$ accepts L. Since the heads can see each other, we can assume that they do not change their relative positions. Therefore we can simply use h_1, h_2, and h_3 to denote the leading head (rightmost), the second head, and the last head, respectively. Fix a long enough random string X of length n^2+n, where $n \gg 2^{|Q|}$ is large enough so that all subsequent formulae make sense. Let $X = xaa_1 \cdots a_{n-1}$ such that for all i: $|x| = |a| = |a_i| = n$. Further divide x equally: $x = x_1x_2x_3x_4$. We will use x (of length n) as the pattern of the input and will use x, and the a's to construct the text part. We will show that $K(X) < |X|$ for a contradiction. Our input to M, $I = x\#y$, will have length at most $O(n^3)$; hence $\log|I| = O(\log n)$. We always assume that we are in the process of running M on input $x \# y$, where y is determined adaptively depending on the behavior of the machine.

DEFINITION. Let y and z be any two segments in the input I of M. We say that M checked y *with* z if on input I, there is a time such that M moves one head at least one step in z while another head is in y.

CHECKING LEMMA. *Let M accept input $I = x \# uxv$ and x_i of x in uxv is not checked with any other occurrence of x_i in I. Then there is an x_i' such that*

$$K(x_i' \mid x - x_i, u - x_i, v - x_i) \leqslant O(|M| + \log|I|) \qquad (*)$$

and M also accepts input $x \# ux'v$, where x' is obtained from x by replacing x_i with x_i'.

Remark on how to use the checking lemma. On input $I = x \# uxv$, if x_i is not checked and $K(x_i \mid x - x_i, u - x_i, v - x_i) \gg O(|M| + \log|I|)$, then M also accepts input $x \# ux'v$, where $x' \neq x$ because $K(x_i) \gg K(x_i') = O(\log n)$. Hence M accepts a wrong input. Later, we will simply refer to the checking lemma without repeating this argument. It is important to note the requirement that $uxv - x_i$ must not contain much information about x_i.

Proof. Define a crossing sequence at a position p of the input to be the sequence of *ID*'s, ordered by time, where each *ID* contains the following information of M

(location of h_1, location of h_2, location of h_3, current state)

at the steps when some head enters position p for the first time. Each *ID* needs total description length at most $O(|M| + \log|I|)$. For M, a crossing sequence contains three *ID*'s. Let |c.s.| denote the description length of a crossing sequence, then $|\text{c.s.}| \leqslant O(|M| + \log|I|)$. Let c.s._1 and c.s._2 be the crossing sequence at the two positions p_1 and p_2 surrounding (one bit before and one bit after) the segment x_i,

respectively. We search for an x_i' as follows. We enumerate each string $\bar{x}_i$ of length $|x_i|$. We replace x_i of x in I with $\bar{x}_i$. We partially simulate M. Start M with status described by the first *ID* in c.s.$_1$ at position p_1. At this point a head enters $\bar{x}_i$. Simulate M honestly. During the simulation, if some head attempts to get into an x_i segment in x, u, or v, then disregard this $\bar{x}_i$.

Note that it is possible that some head is *stationary* in some other x_i segment, but then by definition such head does not move since otherwise our x_i is checked with some other occurrence of x_i. So we just need to remember that bit (or two bits for two heads) in order to perform the simulation for the pass.

When the head reaches p_2, we check if the current status of the machine matches the description of the corresponding *ID* in c.s.$_2$. If not, disregard this $\bar{x}_i$. If consistent, take the next *ID* from c.s.$_1$ and repeat above. We repeat this three times for three heads. The above simulation can be carried out only with information $x - x_i$, $u - x_i$, $v - x_i$, c.s.$_1$, c.s.$_2$, and a description of M. We are guaranteed to find such an x_i' since we know at least one exists. Hence the first x_i' we found is determined by the crossing sequences and $uxv - x_i$, so

$$K(x_i' \mid x - x_i, u - x_i, v - x_i) \leqslant O(|M| + \log|I|). \qquad \text{Q.E.D.}$$

MOVING LEMMA. *Let a be a block of length n in* text, *such that $K(a) \geqslant n - O(\log n)$. Assume that M is in state q and one of the heads h_i, $1 \leqslant i \leqslant 3$, is at the first bit of a. Then either*

1. *while h_i crosses a, some other head moves at least one step, or*

2. *h_i will move regardless of what it reads until it reaches end of input or meets with some other head, while all other heads stay stationary.*

Proof. If (2) is false, then from each state of M, during the time h_i is passing through a, there is a sequence of at most $|Q|$ long that will make some other head move. This is because that M has only $|Q|$ states and the shortest distance from any state to any other state is at most $|Q|$ if there is a path at all. Now assume that h_i passes a without any other head moving, we will show that a is significantly compressible. We use M to compress a.

At each bit of a, a certain combination of sequence of length $|Q|$ starting from this bit cannot occur since this sequence will lead to the movement of another head. We know that a satisfies the following properties:

1. If M starts with state q and h_i at beginning of a, h_i should move through a without moving other heads.

2. At any step, M is always in a state such that there is a string of length $|Q|$ which would lead to the movement of other heads.

The number of strings of length $|a|$ satisfying above two properties is at most, by simple counting,

$$(2^{|Q|} - 1)^{|a|/|Q|}.$$

We can now code a as the ith string of length $|a|$ such that M starting from state q will move through it without moving other heads. Number i can be described in $(1-\varepsilon)|a|$ bits by the above, where ε is fixed constant depending only on $|Q|$. Other information needed is

- Description of M, $O(1)$ bits;
- The current bits read by other heads, $O(1)$ bits;
- Current state of M, $O(1)$ bits.

Total is less than $K(a)$, a contradiction. Q.E.D.

TWO-HEAD MOVING LEMMA. *Assume that* $K(abcd) \geqslant |abcd| - O(\log n)$ *and* $|a|$, $|b|$, $|c|$, $|d| > n/8$, *and assume that* h_1 *is at the beginning of* a *and* h_2 *is at the beginning of* b *on input* $x\# \cdots bc \cdots ad \cdots$. *Then when* h_1 *finishes* a *or* h_2 *finishes* b, *either*

1. h_3 *has moved at least one step; or otherwise*

2. h_1 *and* h_2 *will keep on moving alone, no matter what the input is* afterward, *until they meet or one of them reaches the end of input.*

Furthermore, if (2) *is true, then after one of* h_1 *or* h_2 *finishes* c *or* d, *no matter what the input is afterward, either*

(2.1) *There will always be some input strings (of length* $\leqslant |Q|$*) that make* h_1 *and* h_2 *both move before they meet or reach the* \$ *sign; or*

(2.2) *One head will move alone until it reaches the* \$ *sign or meet the other head.*

Proof. The proof is very similar to the (one-head) moving lemma. If (2) is false, then from any current state before h_1 moves out of a or h_2 moves out of b, with some two input segments each of length $|Q|$ under h_1 and h_2, respectively, M will move h_3. This is again because the shortest distance from any current state to any other reachable state in the state diagram of M is at most $|2Q|$. By similar proof as in the moving lemma we can compress ab. String ab satisfies the following two properties:

1. If M starts with state q, h_1 at beginning of a, and h_2 at beginning of b, then h_3 will not move until h_1 reaches the end of a or h_2 reaches the end of b.

2. At any step, M is always in a state such that there are two strings each of length $|Q|$ which, if read by h_1 and h_2, respectively, would lead to the movements of h_3.

The number of strings of length $|ab|$ that satisfy above two properties is at most

$$(2^{2|Q|}-1)^{|ab|/2|Q|}.$$

The rest of the proof for Cases (1) and (2) is similar to that of moving lemma, hence omitted.

The same proof also works for Cases (2.1) and (2.2) except we need the following observation. Let p be a state from which, once entered, M will always stay in some state such that both h_1, h_2 will always have a chance to move with some input combination. Let q be a state such that, once entered, only one head can move until it meets the other or reads \$. If M does not have such states, we will have that case (2.1) is true. Otherwise from any current state of M to p or q, the shortest distance is at most $|Q|$, and we repeat above argument so that one of p and q must be entered (or *abcd* can be greatly compressed). If p is entered first then (2.1) is true. If q is entered first, (2.2) is true. Q.E.D.

We begin to prove the main theorem. All the heads are initially at the \$ sign. One head must first move passing #. We claim that with input

$$x \# a^{2n} x_1 x_2 x_3 \cdots \tag{2}$$

h_2 must pass the # sign when h_1 reaches the end of a^{2n}. If this is not true h_1 would move to the end of input anyway (while h_2, h_3 stay stationary) by the moving lemma. Then on input $x \# a^{2n} x$, h_1 first reaches the end before h_2 reaches # sign. Then when h_2 passes through a^{2n}, again by the moving lemma, either h_2 is out of *pattern* x or h_2 moves to the end anyway (while h_3 stays stationary). In either case x is not checked. Hence when h_1 reaches the end of a^{2n} h_2 is in a^{2n}.

Now, when h_1 finishes reading x_2 of *text*, we consider three possibilities:

1. h_3 has not moved and h_2 has already reached x_2 of *text*. In this case we append x_4 to the input and then x_1 cannot be checked, this leads to contradiction by the checking lemma;

2. h_3 has not moved and h_2 has not reached x_2 of the *text*. Then h_1 and h_2 are at least $n/4$ bits apart. By the two-head moving lemma, either h_3 makes a move before h_1 finishes reading x_3 or h_1 and h_2 move until they meet or h_1 reaches the end of the input (without moving h_3) no matter what they read. In the previous case (h_3 moves), case 3 applies; For the latter case (h_3 does not move), by (2) of the two-head moving lemma, when h_1 finishes reading x_3, either (2.1) or (2.2) of the lemma is true.

If (2.1) is true, then two heads move. We append $x_4 a^m$\$ to the partial input, where $m = nl$ and l is the length of the current partial input. With this input, each time h_1 crosses an a-block, h_2 has to make a move by the two-head moving lemma (case 2.1) and the moving lemma. So when h_1 reaches the end of input, h_2 is out of x_4; hence x_4 is not checked, a contradiction. It is also possible that h_1 and h_2 meet before h_1 reaches the end, but in this case h_2 is out of x_3, and hence x_3 cannot be checked.

If (2.2) is true, then only one head moves. We append $a^m x$\$ to the partial input, where m is same as above. Now h_1 first goes to the end of input alone, while h_2

stays before a^m. Then by the moving lemma, by the time h_2 reaches the end of a^m, either h_3 is out of the pattern $x\#$ or h_2 goes to the end of the input. In either case x_4 in the text cannot be checked.

3. h_3 makes a move. In this case, we append a_1^{2n} to the partial input to obtain

$$x \# a^{2n} x_1 x_2 x_3 a_1^{2n}. \tag{3}$$

When h_1 finishes reading a_1^{2n}, by the two-head moving lemma, we can have the following cases:

(I) h_3 is out of the *pattern.*

(II) h_2 entered a_1^{2n};

(III) h_3 is in the pattern, h_2 has not entered a_1^{2n}, and h_1 moves alone to the end;

(IV) h_3 is in the pattern, h_2 has not entered a_1^{2n}, and h_1 and h_2 will move until they meet or one reaches \$.

In case (I) we append x to the input. Then the x_4 part of this x cannot be checked and we are done.

In case (II), we add $a_2^{2n} x_1 x_2 x_3$ to the input and repeat the above argument so that each time h_3 is moved at least one step. Eventually h_3 is out of the *pattern* or some of the other previous cases happen, so we are done. In each iteration, we use the next a block. Then length of the whole text is bound by some polynomial in n (less than $O(n^3)$).

In case (III), append x to the input. Then by the moving lemma when h_2 passes a_1^{2n}, h_3 either moves at least one step per a_1-block or it stays stationary until h_2 reaches \$, and in either case x_4 in x cannot be checked.

In case (IV) we also append x to input. Then since both heads are moving and (II) is not true; when h_1 reaches \$, h_2 has not reached the middle of $a_1^{2n}x$ yet, by the moving lemma. Then the argument is the same as in case (III).

Note that the *text* is constructed solely by $a-s$, x_1, x_2, x_3, and $K(x_4 \mid \text{text}) \ll |x_4|$. This makes the checking lemma applicable. We have completed our proof. Q.E.D.

ACKNOWLEDGMENTS

The authors thank Athanasios Tsantilas for his numerous improvements on the manuscript and Joel Seiferas, Zvi Galil, and Yaacov Yesha for many discussions on this topic. We also thank John Tromp for proofreading the paper and a referee for pointing out several errors in the paper. Part of this work was done while both authors visited Technion, Haifa. We are grateful for the hospitality of the computer science and the electrical engineering departments, especially Benny Chor and Amos Israeli.

REFERENCES

[BM] R. S. BOYER AND J. S. MOORE, A fast string searching algorithm, *Comm. ACM* **20**, No. 10 (1977), 762–772.

[C] S. COOK, Linear time simulation of deterministic two-way pushdown automata, *in* "Proceedings, IFIP Congress 71," pp. 172–179, TA-2, North-Holland, Amsterdam, 1971.

[DG] P. DURIS AND Z. GALIL, Fooling a two-way automaton or one pushdown store is better than one counter for two way machines, *Theoret. Comput. Sci.* **21** (1982), 39–53.

[G] Z. GALIL, Open problems in stringology, *in* "Combinatorial Algorithms in Words" (A. Apostolico and Z. Galil, Eds.), NATO ASI Ser. F, No. 12, pp. 1–8, Springer-Verlag, New York/Berlin, 1985.

[GS] Z. GALIL AND J. SEIFERAS, Time-space-optimal string matching, *in* "Proceedings, 13th ACM Symp. on Theory of Computing, 1981," pp. 106–113.

[KMP] D. E. KNUTH, J. H. MORRIS, JR., AND V. R. PRATT, Fast pattern matching in strings, *SIAM J. Comput.* **6**, No. 2 (1977), 323–350.

[L] M. LI, Lower bounds on string-matching, manuscript, 1985.

[LY] M. LI AND Y. YESHA, String-matching cannot be done by a two-head one-way deterministic finite automaton, *Inform. Process. Lett.* **22** (1986), 231–235.

[LV] M. LI AND P. VITANYI, Two decades of Kolmogorov complexity, *in* "Proceedings, IEEE 3rd Structure in Complexity Theory Conference, 1988," pp. 80–101.

[LV1] M. LI AND P. VITANYI, Tape versus queue and stacks: The lower bounds, *Inform. and Comput.* **78**, No. 1 (1988), 56–85.

[M] W. MAASS, Combinatorial lower bound arguments for deterministic and nondeterministic Turing machines, *Trans. Amer. Math. Soc.* **292**, 675–693.

[P] W. PAUL, Kolmogorov complexity and lower bounds, *in* "2nd International Conference on Fundamentals of Computation Theory, 1979."

[PSS] W. PAUL, J. SEIFERAS, AND J. SIMON, An information theoretic approach to time bounds for on-line computations, *J. Comput. System. Sci.* **23** (1981), 108–126.

[RS] S. REISCH AND G. SCHNITGER, Three applications of Kolmogorov-complexity, *in* "Proceedings, 23rd IEEE Symp. on Foundations of Computer Science, 1982," pp. 45–52.